

Goal: Creating a complete structure of classes and functions for an image editing API in Dart, similar to software like Photoshop, is a complex task that would require significant planning and development effort. Such an API would encompass a wide range of features and techniques related to image editing and manipulation. Here's a high-level outline of some known features and techniques you might consider implementing in your API: use json serialisation to manage each state.

Image Basics:

- Loading and saving images in various formats (e.g., JPEG, PNG, BMP).

- Creating new images with specified dimensions and backgrounds.

- Resizing and cropping images.

- Rotating and flipping images.

Layers and Compositing:

- Support for multiple layers in an image.

- Blending modes for layer compositing (e.g., normal, multiply, overlay).

- Layer opacity and visibility control.

- Layer grouping and organization.

Filters and Effects:

- Applying filters and effects to images and layers.

- Common filters like blur, sharpen, brightness, contrast, and saturation.

- Customizable filter parameters.

Selections and Masks:

- Creating and manipulating selections.

- Feathering and antialiasing selections.

- Applying masks to layers and selections.

Drawing and Painting:

- Drawing shapes (e.g., lines, rectangles, circles).

- Brush tools for freehand painting.

- Text tools for adding and editing text on images.

- Gradient fills and pattern fills.

Transformations:

- Distortions and perspective transformations.

- Warping and liquify tools.

- Mesh transformations for complex distortions.

Undo and Redo:

- Implementing an undo/redo system for reverting changes.

- Storing and managing a history of image states.

Color Correction and Adjustment:

- Histogram equalization.

- Color balance and color correction.

- Levels and curves adjustments.

---

Adding Task: Define the main class structure:

---

Adding Task: - Create a class named "Image" that represents an image and contains properties such as width, height, and pixel data.

---

Adding Task: - Create a class named "Layer" that represents a layer in the image and contains properties such as opacity, visibility, and blending mode.

---

Adding Task: - Create a class named "Filter" that represents a filter or effect that can be applied to an image or layer.

---

Adding Task: - Create a class named "Selection" that represents a selection area in the image or layer.

---

Adding Task: - Create a class named "Brush" that represents a brush tool for freehand painting.

---

Adding Task: - Create a class named "Text" that represents a text tool for adding and editing text on images.

---

Adding Task: - Create a class named "Transformation" that represents a transformation operation on an image or layer.

---

Adding Task: - Create a class named "History" that manages the history of image states for undo/redo functionality.

---

Adding Task: - Create a class named "ColorCorrection" that represents color correction and adjustment operations on an image or layer.

---

Adding Task: Implement the basic image editing functions:

---

Adding Task: - Implement functions for loading and saving images in various formats.

---

Adding Task: - Implement functions for creating new images with specified dimensions and backgrounds.

---

Adding Task: - Implement functions for resizing and cropping images.

---

Adding Task: - Implement functions for rotating and flipping images.

---

Adding Task: Implement the layers and compositing functionality:

---

Adding Task: - Implement support for multiple layers in an image.

---

Adding Task: - Implement blending modes for layer compositing.

---

Adding Task: - Implement layer opacity and visibility control.

---

Adding Task: - Implement layer grouping and organization.

---

Adding Task: Implement the filters and effects functionality:

---

Adding Task: - Implement functions for applying filters and effects to images and layers.

---

Adding Task: - Implement common filters like blur, sharpen, brightness, contrast, and saturation.

---

Adding Task: - Implement customizable filter parameters.

---

Adding Task: Implement the selections and masks functionality:

---

Adding Task: - Implement functions for creating and manipulating selections.

---

Adding Task: - Implement functions for feathering and antialiasing selections.

---

Adding Task: - Implement functions for applying masks to layers and selections.

---

Adding Task: Implement the drawing and painting functionality:

---

Adding Task: - Implement functions for drawing shapes like lines, rectangles, and circles.

---

Adding Task: - Implement brush tools for freehand painting.

---

Adding Task: - Implement text tools for adding and editing text on images.

---

Adding Task: - Implement gradient fills and pattern fills.

---

Adding Task: Implement the transformations functionality:

---

Adding Task: - Implement functions for distortions and perspective transformations.

---

Adding Task: - Implement warping and liquify tools.

---

Adding Task: - Implement mesh transformations

---

Starting task: Define the main class structure:

---

Generating response...

---

Executing: "Define the main class structure:": To create a complete structure of classes and functions for an image editing API in Dart, similar to software like Photoshop, we can start by defining the main class structure. Here's an outline of the main classes and their relationships:

1. ImageEditingAPI Class:

- This is the main class that will serve as the entry point for using the API.
- It will contain methods for loading and saving images, creating new images, and managing the history of image

states.

## 2. Image Class:

- This class represents an image and will contain the pixel data, dimensions, and other properties.
- It will have methods for resizing, cropping, rotating, and flipping the image.

## 3. Layer Class:

- This class represents a layer in an image and will contain the pixel data, blending mode, opacity, and visibility properties.
- It will have methods for applying filters and effects, as well as managing selections and masks.

## 4. Selection Class:

- This class represents a selection on a layer.
- It will have methods for creating and manipulating selections, as well as applying masks.

## 5. DrawingTool Class:

- This class represents a tool for drawing shapes, such as lines, rectangles, and circles.
- It will have methods for drawing shapes on layers.

## 6. BrushTool Class:

- This class represents a tool for freehand painting.
- It will have methods for painting on layers.

## 7. TextTool Class:

- This class represents a tool for adding and editing text on images.
- It will have methods for adding and editing text on layers.

## 8. FillTool Class:

- This class represents a tool for filling shapes with gradients or patterns.
- It will have methods for filling shapes on layers.

## 9. TransformationTool Class:

- This class represents a tool for applying transformations, such as distortions, perspective transformations, and mesh transformations.
- It will have methods for applying transformations on layers.

## 10. UndoRedoManager Class:

- This class manages the undo/redo functionality for reverting changes.
- It will store and manage a history of image states using a stack data structure.

## 11. ColorCorrection Class:

- This class represents color correction and adjustment functionalities.
- It will have methods for histogram equalization, color balance, and levels and curves adjustments.

Each of these classes can be further expanded with additional methods and properties based on specific requirements. The relationships between these classes will allow for a comprehensive set of image editing functionalities in the API.

---

Adding Task: Define the main class structure:

---

Adding Task: Create a class named "BlendingMode" that represents the blending modes for layer compositing.

---

Starting task: - Create a class named "Image" that represents an image and contains properties such as width, height, and pixel data.

---

Writing code...

---

Executing: "- Create a class named "Image" that represents an image and contains properties such as width, height, and pixel data.": ``dart  
// Create a class named "Image" that represents an image and contains properties such as width, height, and pixel data.

```
class Image {  
  int width; // Width of the image  
  int height; // Height of the image  
  List<List<int>> pixelData; // Pixel data of the image  
  
  // Constructor to initialize the image  
  Image(int width, int height) {  
    this.width = width;  
    this.height = height;  
    pixelData = List<List<int>>.generate(height, (_) => List<int>.generate(width, (_) => 0));  
  }  
}
```

The code defines a class named "Image" that represents an image. It contains three properties: width, height, and pixelData. The width and height properties represent the dimensions of the image, while the pixelData property is a 2D array that holds the pixel values of the image. The constructor takes in the width and height as parameters and initializes the pixelData array with zeros.

---

Adding Task: - Create a class named "Image" that represents an image and contains properties such as width, height, and pixel data.

---

Starting task: - Create a class named "Layer" that represents a layer in the image and contains properties such as opacity, visibility, and blending mode.

---

Writing code...

---

Executing: "- Create a class named "Layer" that represents a layer in the image and contains properties such as opacity, visibility, and blending mode.": ``dart  
// Create a class named "Layer" that represents a layer in the image  
class Layer {  
 // Properties of the layer  
 double opacity;  
 bool isVisible;  
 String blendingMode;  
  
 // Constructor

```
Layer({this.opacity = 1.0, this.isVisible = true, this.blendingMode = 'normal'});  
}  
...
```

Explanation:

- This code defines a class named "Layer" with properties such as opacity, visibility, and blending mode.
- The `opacity` property represents the transparency of the layer, with a default value of 1.0 (fully opaque).
- The `isVisible` property indicates whether the layer is visible or not, with a default value of `true`.
- The `blendingMode` property specifies the blending mode to be applied when compositing the layer with other layers, with a default value of 'normal'.
- The class also includes a constructor that allows you to initialize the properties when creating a new instance of the Layer class.

---

Adding Task: - Create a class named "Layer" that represents a layer in the image and contains properties such as opacity, visibility, and blending mode.

---

Adding Task: Create a class named "Image" that represents an image and contains properties such as width, height, and pixel data.

---

Starting task: - Create a class named "Filter" that represents a filter or effect that can be applied to an image or layer.

---

Writing code...

---

Executing: "- Create a class named "Filter" that represents a filter or effect that can be applied to an image or layer.":

```
```dart  
class Filter {  
  // Properties  
  String name;  
  Map<String, dynamic> parameters;  
  
  // Constructor  
  Filter({required this.name, required this.parameters});  
  
  // Methods  
  void apply() {  
    // Apply the filter to the image or layer using the specified parameters  
    // ...  
  }  
  
  void updateParameters(Map<String, dynamic> newParameters) {  
    // Update the filter parameters with the new values  
    parameters = newParameters;  
  }  
  
  // Other methods specific to each filter can be added here  
}  
  
// Example usage:
```

```
Filter blurFilter = Filter(name: 'Blur', parameters: {'radius': 5});
blurFilter.apply();
```

```
Filter brightnessFilter = Filter(name: 'Brightness', parameters: {'level': 0.5});
brightnessFilter.updateParameters({'level': 0.8});
brightnessFilter.apply();
...
```

The `Filter` class represents a filter or effect that can be applied to an image or layer. It has a `name` property to identify the filter and a `parameters` property to store the specific parameters required by the filter. The `apply()` method applies the filter to the image or layer using the specified parameters. The `updateParameters()` method allows updating the filter parameters with new values. Additional methods specific to each filter can be added as needed.

---

Adding Task: - Create a class named "Filter" that represents a filter or effect that can be applied to an image or layer.

---

Adding Task: Create a class named "ImageEditor" that represents the main interface for the image editing API. This class should contain functions for each of the implemented image editing features, such as loading and saving images, resizing and cropping images, applying filters and effects, creating and manipulating selections, drawing shapes, and performing transformations.

---

Starting task: - Create a class named "Selection" that represents a selection area in the image or layer.

---

Generating response...

---

Executing: "- Create a class named "Selection" that represents a selection area in the image or layer.": To complete the task of creating a class named "Selection" that represents a selection area in the image or layer, I would implement the following code in Dart:

```
dart
class Selection {
  int x; // x-coordinate of the top-left corner of the selection area
  int y; // y-coordinate of the top-left corner of the selection area
  int width; // width of the selection area
  int height; // height of the selection area

  Selection(this.x, this.y, this.width, this.height);

  void move(int dx, int dy) {
    // Update the position of the selection area by applying the given delta values
    x += dx;
    y += dy;
  }

  void resize(int dw, int dh) {
    // Update the dimensions of the selection area by applying the given delta values
    width += dw;
    height += dh;
  }
}
```

```

bool containsPoint(int px, int py) {
// Check if the given point (px, py) is within the selection area
return px >= x && px < x + width && py >= y && py < y + height;
}
}
...

```

In the above code, I have created a class named "Selection" that represents a selection area in the image or layer. It has four properties: `x` and `y` to store the coordinates of the top-left corner of the selection area, and `width` and `height` to store the dimensions of the selection area.

The constructor of the class takes the initial `x`, `y`, `width`, and `height` values and initializes the corresponding properties.

The `move` method allows the selection area to be moved by applying the given delta values (`dx` and `dy`) to the current coordinates.

The `resize` method allows the dimensions of the selection area to be changed by applying the given delta values (`dw` and `dh`) to the current dimensions.

The `containsPoint` method checks if a given point (`px`, `py`) is within the selection area by comparing its coordinates with the coordinates of the top-left corner and the dimensions of the selection area.

This implementation provides a basic representation of a selection area and allows for movement, resizing, and checking if a point is within the selection area.

---

Adding Task: - Create a class named "Selection" that represents a selection area in the image or layer.

---

Adding Task: Create a class named "Filter" that represents a filter operation on an image or layer.

---

Starting task: - Create a class named "Brush" that represents a brush tool for freehand painting.

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Summarizing Creating a complete structure of classes and functions for an image editing API in Dart, similar to software like Photoshop, is a complex task that would require significant planning and development effort. Such an API would encompass a wide range of features and techniques related to image editing and manipulation. Here's a high-level outline of some known features and techniques you might consider implementing in your API: use json serialisation to manage each state.

Loading and saving images in various formats (e.g., JPEG, PNG, BMP).

Creating new images with specified dimensions and backgrounds.

Resizing and cropping images.

Rotating and flipping images.Layers and Compositing:

Support for multiple layers in an image.

Blending modes for layer compositing (e.g., normal, multiply, overlay).

Layer opacity and visibility control.

Layer grouping and organization.



## Filters and Effects:

Applying filters and effects to images and layers.

Common filters like blur, sharpen, brightness, contrast, and saturation.

Customizable filter parameters.

## Selections and Masks:

Creating and manipulating selections.

Feathering and antialiasing selections.

Applying masks to layers and selections.

## Drawing and Painting:

Drawing shapes (e.g., lines, rectangles, circles).

Brush tools for freehand painting.

Text tools for adding and editing text on images.

Gradient fills and pattern fills.

## Transformations:

Distortions and perspective transformations.

Warping and liquify tools.

Mesh transformations for complex distortions.

## Undo and Redo:

Implementing an undo/redo system for reverting changes.

Storing and managing a history of image states.

## Color Correction and Adjustment:

Histogram equalization.

Color balance and color correction.

Levels and curves adjustments."": Creating a complete structure of classes and functions for an image editing API in Dart, similar to software like Photoshop, is a complex task that would require significant planning and development effort. Such an API would encompass a wide range of features and techniques related to image editing and manipulation. Here's a high-level outline of some known features and techniques you might consider implementing in your API:

## Image Basics:

- Loading and saving images in various formats (e.g., JPEG, PNG, BMP).
- Creating new images with specified dimensions and backgrounds.
- Resizing and cropping images.
- Rotating and flipping images.

## Layers and Compositing:

- Support for multiple layers in an image.
- Blending modes for layer compositing (e.g., normal, multiply, overlay).
- Layer opacity and visibility control.
- Layer grouping and organization.

## Filters and Effects:

- Applying filters and effects to images and layers.
- Common filters like blur, sharpen, brightness, contrast, and saturation.

- Customizable filter parameters.

#### Selections and Masks:

- Creating and manipulating selections.
- Feathering and antialiasing selections.
- Applying masks to layers and selections.

#### Drawing and Painting:

- Drawing shapes (e.g., lines, rectangles, circles).
- Brush tools for freehand painting.
- Text tools for adding and editing text on images.
- Gradient fills and pattern fills.

#### Transformations:

- Distortions and perspective transformations.
- Warping and liquify tools.
- Mesh transformations for complex distortions.

#### Undo and Redo:

- Implementing an undo/redo system for reverting changes.
- Storing and managing a history of image states.

#### Color Correction and Adjustment:

- Histogram equalization.
- Color balance and color correction.
- Levels and curves adjustments.

This outline provides a comprehensive overview of the key features and techniques that could be included in your image editing API. By implementing these functionalities using classes and functions, you can create a powerful and flexible tool for image editing in Dart similar to software like Photoshop.

---

Executing: "Response for 'continue to generate'": Certainly! Here's the continuation of the code to generate the remaining classes:

```
```dart
class DrawingTool {
  // Properties and methods specific to the DrawingTool class can be added here
}

class BrushTool {
  // Properties and methods specific to the BrushTool class can be added here
}

class TextTool {
  // Properties and methods specific to the TextTool class can be added here
}

class FillTool {
  // Properties and methods specific to the FillTool class can be added here
}
```

```

class TransformationTool {
// Properties and methods specific to the TransformationTool class can be added here
}

class UndoRedoManager {
  List<Image> imageStates = [];

  void saveImageState(Image image) {
    // Save the current state of the image for undo/redo functionality
    imageStates.add(image);
  }

  void undo() {
    // Revert to the previous image state
    if (imageStates.isNotEmpty) {
      imageStates.removeLast();
    }
  }

  void redo() {
    // Redo the previously undone image state
    // ...
  }
}

class ColorCorrection {
// Properties and methods specific to the ColorCorrection class can be added here
}
...

```

In the provided code, I have added the remaining class declarations: `DrawingTool`, `BrushTool`, `TextTool`, `FillTool`, `TransformationTool`, `UndoRedoManager`, and `ColorCorrection`. Each class represents a specific tool or functionality related to image editing.

For the `UndoRedoManager` class, I have included a `saveImageState` method to save the current state of the image for undo/redo functionality. The `undo` method allows reverting to the previous image state, while the `redo` method allows redoing the previously undone image state.

Please note that the provided code is just a basic structure for each class, and you can further expand each class by adding more properties and methods as per your specific requirements.

---

Executing: "Response for 'continue your task'": To continue the task of creating a class structure for an image editing API in Dart, we can define additional classes and their relationships based on the outline provided earlier. Here's an updated outline:

#### 1. ImageEditingAPI Class:

- This is the main class that serves as the entry point for using the API.
- It contains methods for loading and saving images, creating new images, and managing the history of image states.

## 2. Image Class:

- This class represents an image and contains the pixel data, dimensions, and other properties.
- It has methods for resizing, cropping, rotating, and flipping the image.

## 3. Layer Class:

- This class represents a layer in an image and contains the pixel data, blending mode, opacity, and visibility properties.
- It has methods for applying filters and effects, as well as managing selections and masks.

## 4. Selection Class:

- This class represents a selection area in the image or layer.
- It has methods for moving, resizing, and checking if a point is within the selection area.

## 5. DrawingTool Class:

- This class represents a tool for drawing shapes, such as lines, rectangles, and circles.
- It has methods for drawing shapes on layers.

## 6. BrushTool Class:

- This class represents a tool for freehand painting.
- It has methods for painting on layers.

## 7. TextTool Class:

- This class represents a tool for adding and editing text on images.
- It has methods for adding and editing text on layers.

## 8. FillTool Class:

- This class represents a tool for filling shapes with gradients or patterns.
- It has methods for filling shapes on layers.

## 9. TransformationTool Class:

- This class represents a tool for applying transformations, such as distortions, perspective transformations, and mesh transformations.
- It has methods for applying transformations on layers.

## 10. UndoRedoManager Class:

- This class manages the undo/redo functionality for reverting changes.
- It stores and manages a history of image states using a stack data structure.

## 11. ColorCorrection Class:

- This class represents color correction and adjustment functionalities.
- It has methods for histogram equalization, color balance, and levels and curves adjustments.

The relationships between these classes can be established by passing instances of one class to another as arguments or by using composition and aggregation. By implementing the methods and functionalities within each class, we can create a comprehensive structure for an image editing API in Dart.

---

Executing: "Response for 'continue your task'": Sure! To continue with the task of creating a complete structure of classes and functions for an image editing API in Dart, similar to software like Photoshop, we can define additional classes and their relationships based on the outline provided earlier.

Here's an updated outline with the additional classes and their relationships:

1. ImageEditingAPI Class:

- This is the main class that will serve as the entry point for using the API.
- It will contain methods for loading and saving images, creating new images, and managing the history of image states.
- It will have a property to store the current active image.

2. Image Class:

- This class represents an image and will contain the pixel data, dimensions, and other properties.
- It will have methods for resizing, cropping, rotating, and flipping the image.
- It will have a property to store the layers in the image.

3. Layer Class:

- This class represents a layer in an image and will contain the pixel data, blending mode, opacity, and visibility properties.
- It will have methods for applying filters and effects, as well as managing selections and masks.
- It will have a property to store the selection in the layer.

4. Selection Class:

- This class represents a selection on a layer.
- It will have methods for creating and manipulating selections, as well as applying masks.
- It will have properties to store the position, dimensions, and other properties of the selection area.

5. DrawingTool Class:

- This class represents a tool for drawing shapes, such as lines, rectangles, and circles.
- It will have methods for drawing shapes on layers.

6. BrushTool Class:

- This class represents a tool for freehand painting.
- It will have methods for painting on layers.

7. TextTool Class:

- This class represents a tool for adding and editing text on images.
- It will have methods for adding and editing text on layers.

8. FillTool Class:

- This class represents a tool for filling shapes with gradients or patterns.
- It will have methods for filling shapes on layers.

9. TransformationTool Class:

- This class represents a tool for applying transformations, such as distortions, perspective transformations, and mesh transformations.
- It will have methods for applying transformations on layers.

10. UndoRedoManager Class:

- This class manages the undo/redo functionality for reverting changes.
- It will store and manage a history of image states using a stack data structure.

11. ColorCorrection Class:

---

Executing: "Response for 'continue your task'": To continue creating a class named "Selection" that represents a selection area, here's an expanded implementation in Dart:

```
`` dart
class Selection {
  int x; // x-coordinate of the top-left corner of the selection area
  int y; // y-coordinate of the top-left corner of the selection area
  int width; // width of the selection area
  int height; // height of the selection area
  bool active; // flag to indicate if the selection is active or not

  Selection(this.x, this.y, this.width, this.height, {this.active = false});

  void move(int dx, int dy) {
    // Update the position of the selection area by applying the given delta values
    x += dx;
    y += dy;
  }

  void resize(int dw, int dh) {
    // Update the dimensions of the selection area by applying the given delta values
    width += dw;
    height += dh;
  }

  bool containsPoint(int px, int py) {
    // Check if the given point (px, py) is within the selection area
    return px >= x && px < x + width && py >= y && py < y + height;
  }

  void activate() {
    // Activate the selection
    active = true;
  }

  void deactivate() {
    // Deactivate the selection
    active = false;
  }
}
```

In this expanded implementation, I have added an additional property called `active` to the `Selection` class. The `active` property is a boolean flag that indicates whether the selection is currently active or not. It has a default value of `false`.

I have also added two methods, `activate()` and `deactivate()`, that allow you to programmatically change the active state of the selection. By calling `activate()`, you can activate the selection, and by calling `deactivate()`, you can deactivate it.

These additions provide more flexibility in managing the active state of a selection, which can be useful in scenarios where you need to work with multiple selections simultaneously or toggle the active state based on user interactions.

---

Executing: "Response for 'continue your task'": To continue creating a class named "DrawingTool" that represents a tool for drawing shapes, such as lines, rectangles, and circles, you can implement the following code in Dart:

```
`` dart
class DrawingTool {
  String shape; // The shape to be drawn (e.g., "line", "rectangle", "circle")

  DrawingTool(this.shape);

  void draw(Image image, int startX, int startY, int endX, int endY) {
    switch (shape) {
      case "line":
        drawLine(image, startX, startY, endX, endY);
        break;
      case "rectangle":
        drawRectangle(image, startX, startY, endX, endY);
        break;
      case "circle":
        drawCircle(image, startX, startY, endX, endY);
        break;
      default:
        throw Exception("Invalid shape: $shape");
    }
  }

  void drawLine(Image image, int startX, int startY, int endX, int endY) {
    // Logic to draw a line on the image using the specified start and end coordinates
    // ...
  }

  void drawRectangle(Image image, int startX, int startY, int endX, int endY) {
    // Logic to draw a rectangle on the image using the specified start and end coordinates
    // ...
  }

  void drawCircle(Image image, int startX, int startY, int endX, int endY) {
    // Logic to draw a circle on the image using the specified start and end coordinates
    // ...
  }
}
```

In the above code, I have created a class named "DrawingTool" that represents a tool for drawing shapes. It has a property called `shape` that represents the shape to be drawn (e.g., "line", "rectangle", "circle").

The constructor of the class takes the initial shape value and initializes the `shape` property.

The `draw` method takes an `Image` object and the start and end coordinates of the shape to be drawn. It uses a `switch` statement to determine the shape and calls the appropriate drawing method based on the specified shape.

The `drawLine`, `drawRectangle`, and `drawCircle` methods are placeholders for the actual logic to draw the respective shapes on the image. You can implement the specific drawing algorithms according to your requirements.

This implementation allows the `DrawingTool` class to handle different shapes and delegate the drawing logic to the appropriate

---

Executing: "Response for 'continue your task'": Loading...

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Response for 'continue your task'": Loading...

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Response for 'continue your task'": Loading...

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Response for 'continue your task'": Loading...

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Response for 'continue your task after DrawingTool'": Loading...

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Response for 'continue your task after DrawingTool'": Loading...

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Response for 'continue your task after DrawingTool'": Loading...

---

Max loops of 5 exceeded, shutting down.

---

Executing: "Response for 'continue your task after DrawingTool'": Loading...

---

Max loops of 5 exceeded, shutting down.

---